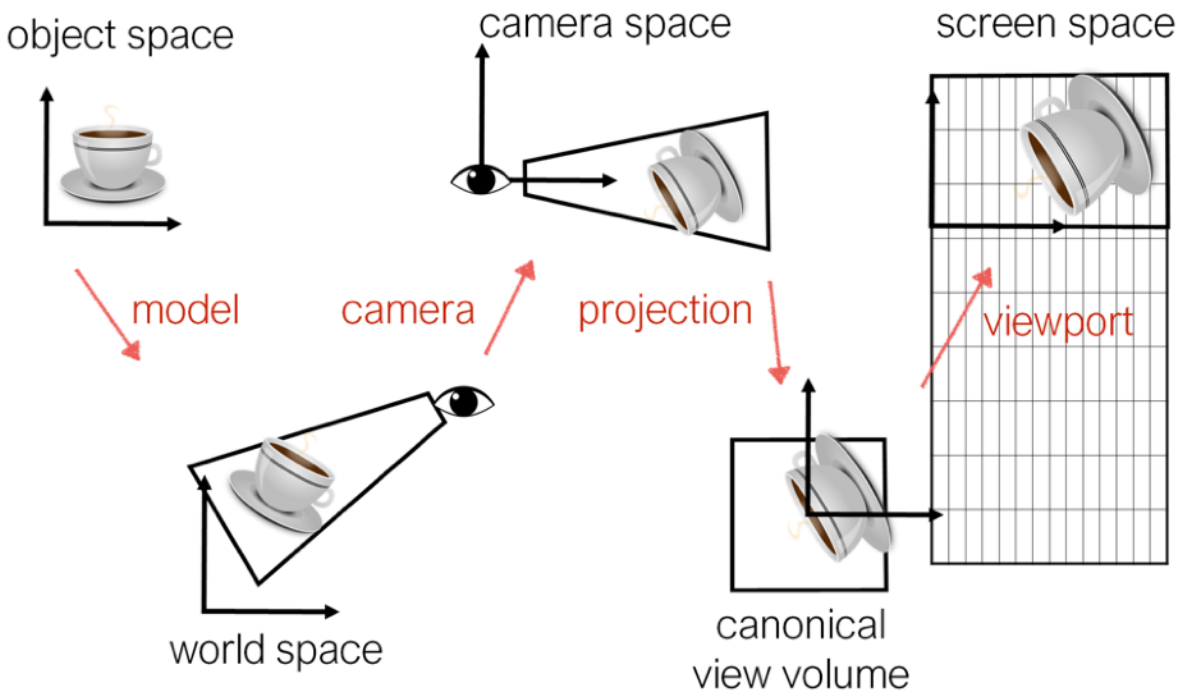
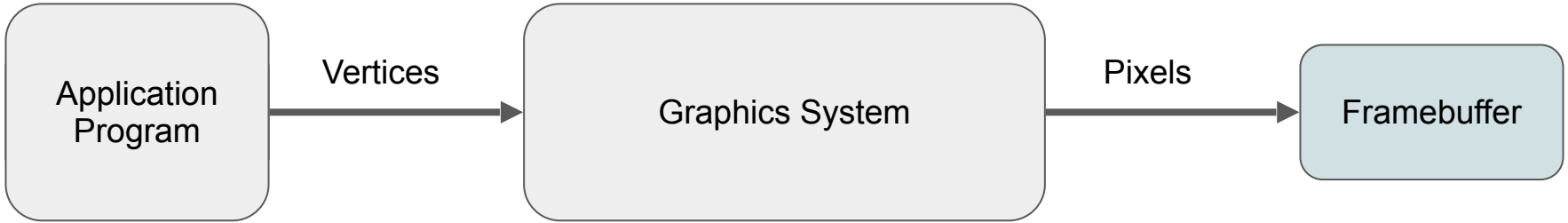
Rendering Pipeline 1

CS 425: Computer Graphics 1

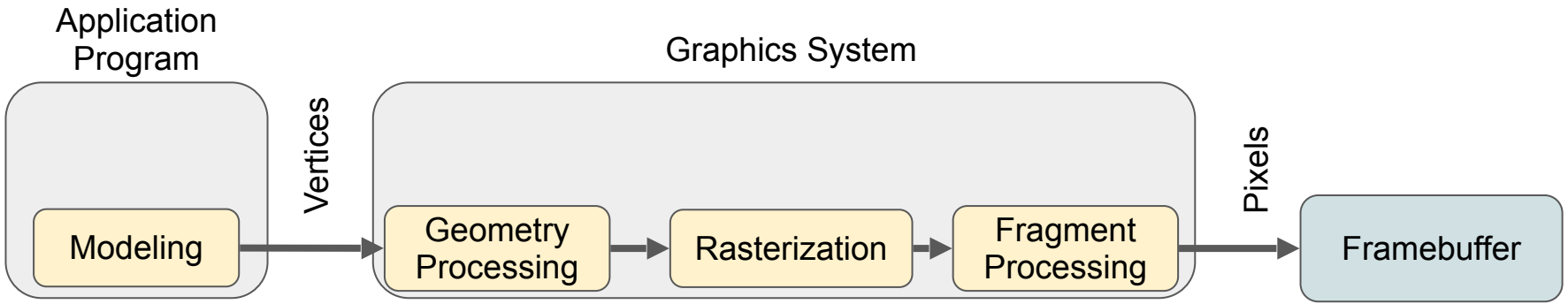
Recap: Transformations



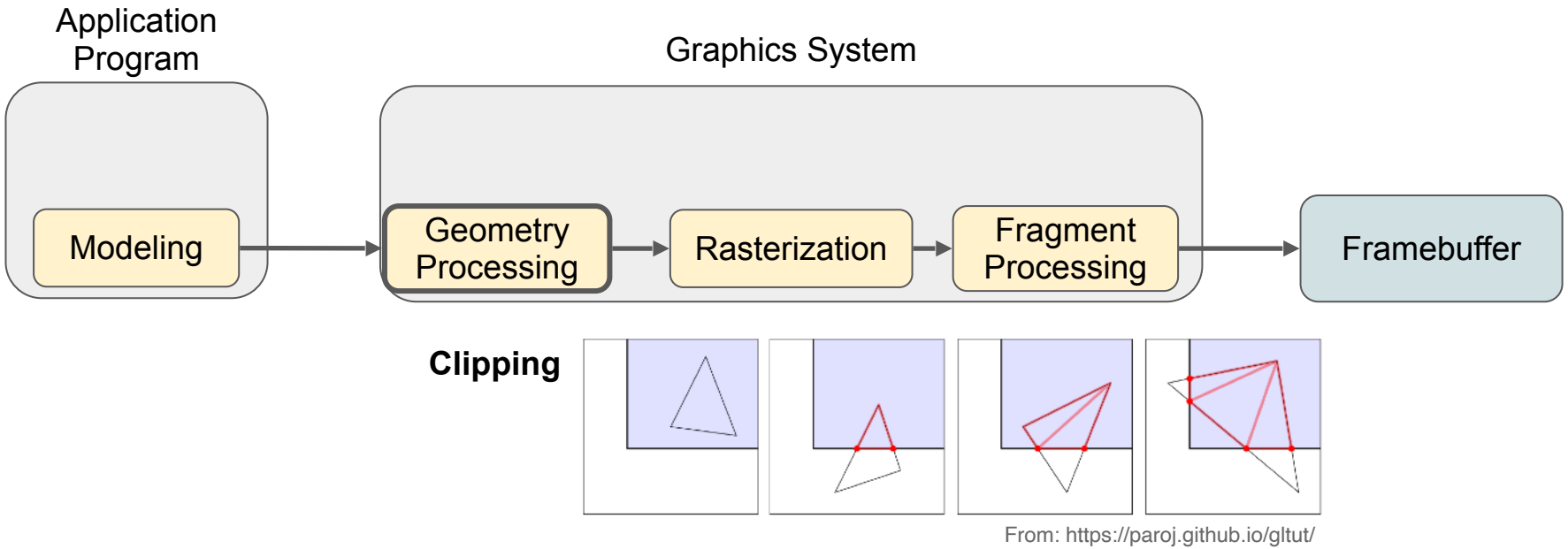
Graphics System



Graphics Pipeline



Clipping



From: https://paroj.github.io/gltut/

Geometry Processing

- View transformations
- Projection
- Primitive assembly
- **Clipping**
- Shading

Geometry Processing

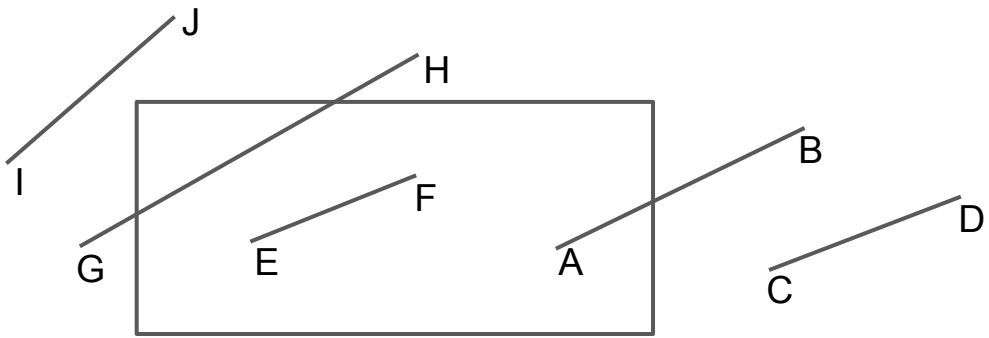
- Projection:
  - Transforming to a normalized view volume.
  - View volume will be a cube of size 2, centered at origin.
  - Eliminates dependency on application specifications as well as display device specifications.
- Primitive Assembly:
  - Grouping of vertices to form objects (primitives).
  - Done before clipping.
- **Clipping**
- Shading
  - Assignment of colors to vertices and determining how the intermediate vertices are to be colored.

Why clipping?

- Rasterization is expensive.
  - Optimize what needs to be rasterized.
  - Clipping: removes geometry and primitives that will not appear on the screen
- Brute force approach: Clip every primitive against all the four sides of the view window (or six sides of the view volume)
- Computing intersections can be expensive since these calculations involve floating point divisions.

Clipping: Line segments

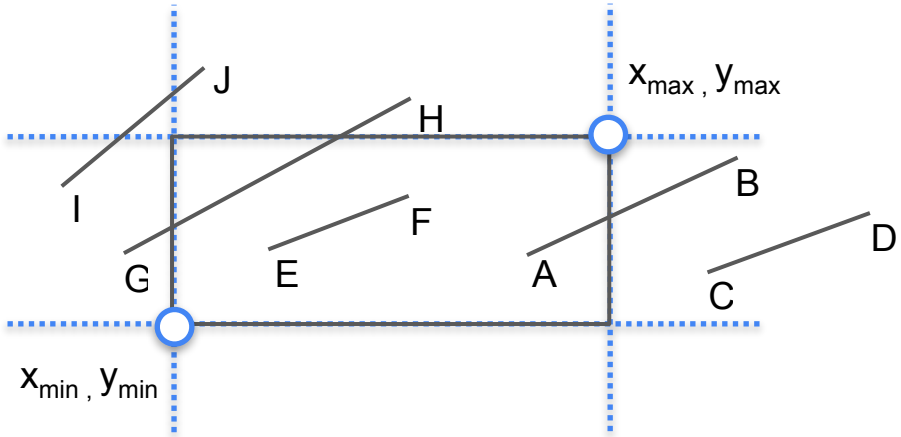
Cohen-Sutherland Line Clipping



Clipping: Cohen-Sutherland Line Clipping

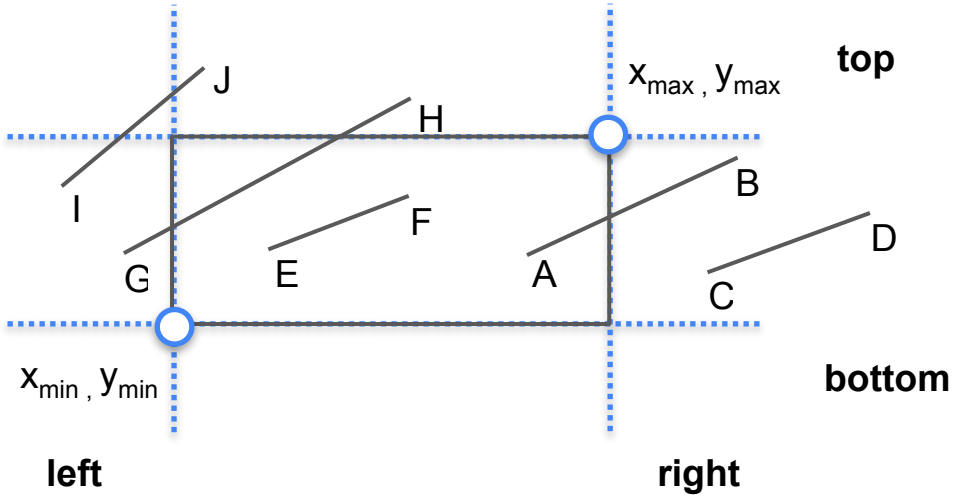
	left	central	right
top	1001	1000	1010
central	0001	0000	0010
bottom	0101	0100	0110

y_{max} separates top from central; y_{min} separates central from bottom; x_{min} separates left from central; x_{max} separates central from right.



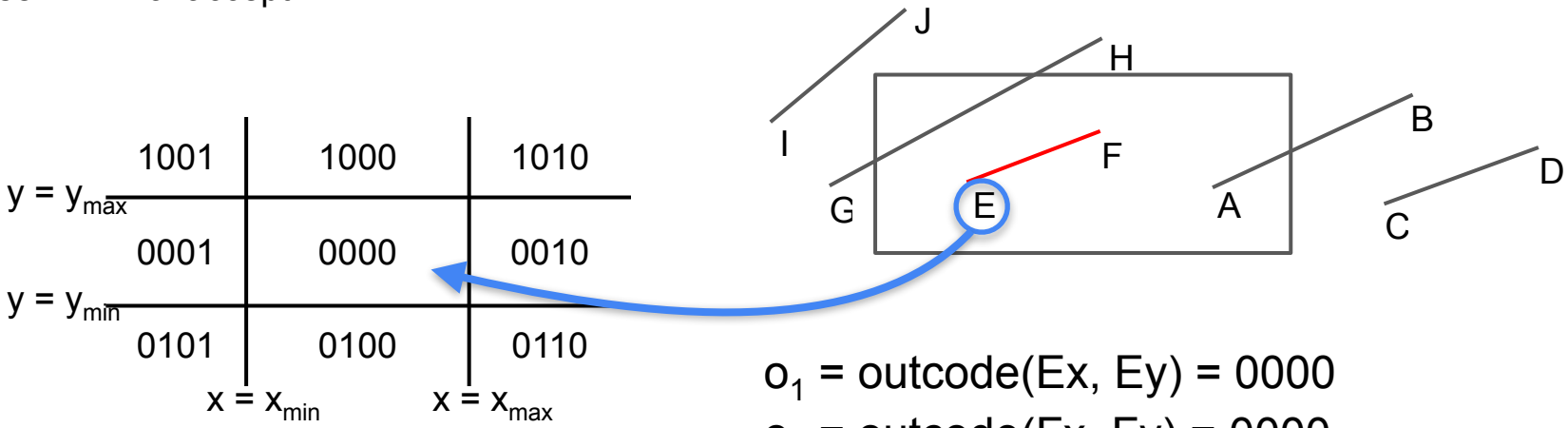
Clipping: Cohen-Sutherland Line Clipping

```
outcode(x, y):
        out = 0000                      inside
    if (y > y_max)
        out = out | 1000                top
    else if (y < y_min)
        out = out | 0100                bottom
    if (x > x_max)
        out = out | 0010                right
    else if (x < x_min)
        out = out | 0001                left
    return out
```



Clipping: Cohen-Sutherland Line Clipping

Case 1: Trivial accept



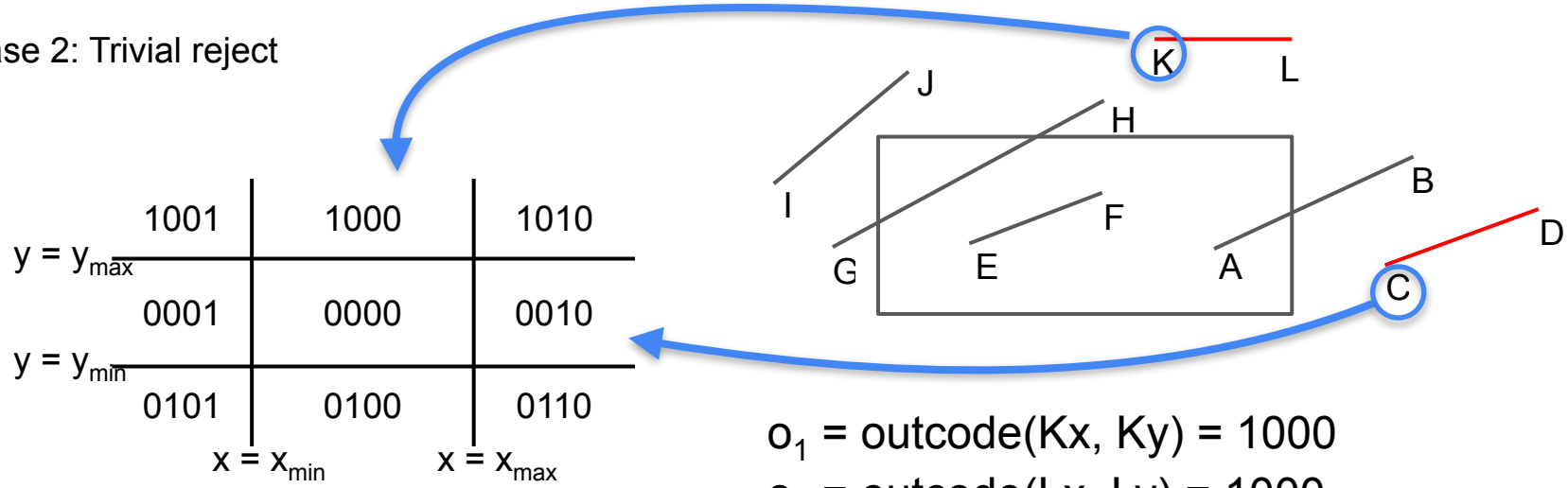
o_1 = outcode(Ex, Ey) = 0000

o_2 = outcode(Fx, Fy) = 0000

$o_1 = o_2 = 0$

Clipping: Cohen-Sutherland Line Clipping

Case 2: Trivial reject



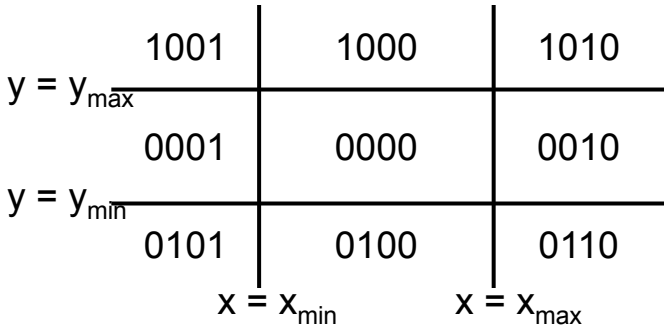
o_1 = outcode(Kx, Ky) = 1000

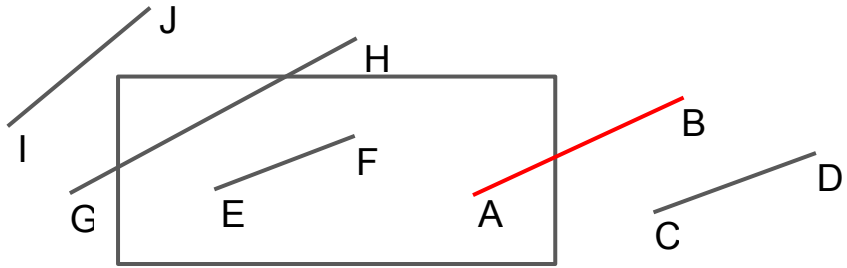
o_2 = outcode(Lx, Ly) = 1000

o_1 & o_2 =1000 ≠ 0

Clipping: Cohen-Sutherland Line Clipping

Case 3:





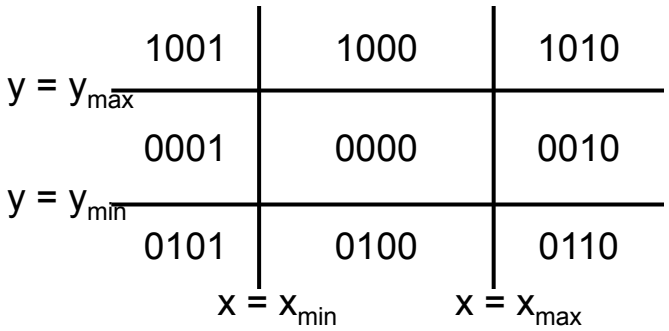
o_1 = outcode(Ax, Ay) = 0000

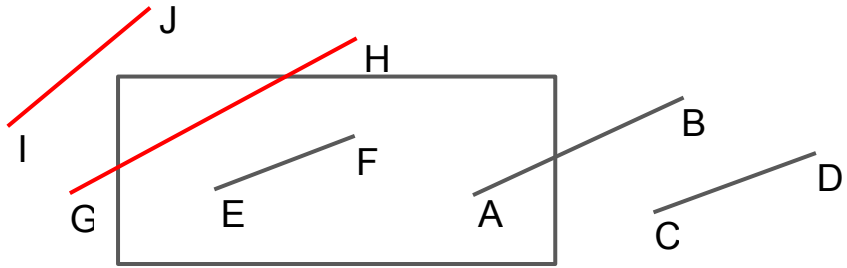
o_2 = outcode(Bx, By) = 0010

$o_1 \neq 0$ and $o_2 = 0$ or vice versa

Clipping: Cohen-Sutherland Line Clipping

Case 4:



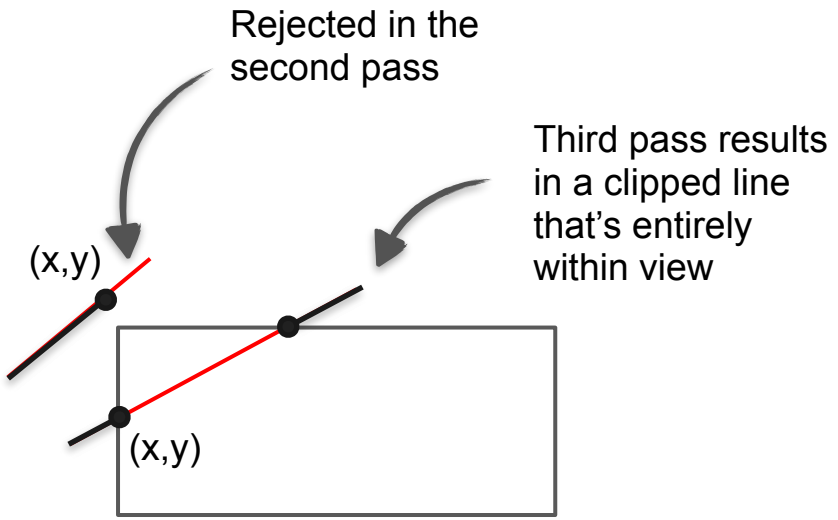


o_1 = outcode(x_1, y_1)

o_2 = outcode(x_2, y_2)

o_1 & o_2 = 0

Clipping: Cohen-Sutherland Line Clipping



Intersection calculation:

- $y = mx + h$
- $m = (y_2 - y_1) / (x_2 - x_1)$
- $(y - y_1) = ((y_2 - y_1) / (x_2 - x_1)) * (x - x_1)$

Disadvantages:

1. Algorithm is recursively applied (till the clipped result can be accepted or rejected).

Clipping: Liang-Barsky Line Clipping

Algorithm makes use of parametric form of line.

Suppose:

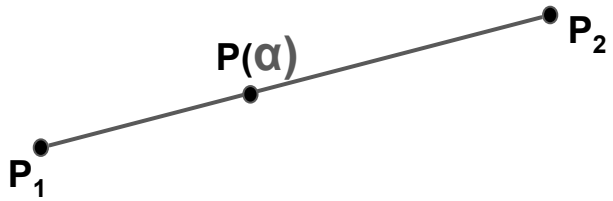
$$P_1 = [x_1, y_1]^T \text{ and } P_2 = [x_2, y_2]^T.$$

Then,

$$P(\alpha) = (1-\alpha)P_1 + \alpha P_2$$

Or

$$x(\alpha) = (1-\alpha)x_1 + \alpha x_2$$
$$y(\alpha) = (1-\alpha)y_1 + \alpha y_2$$



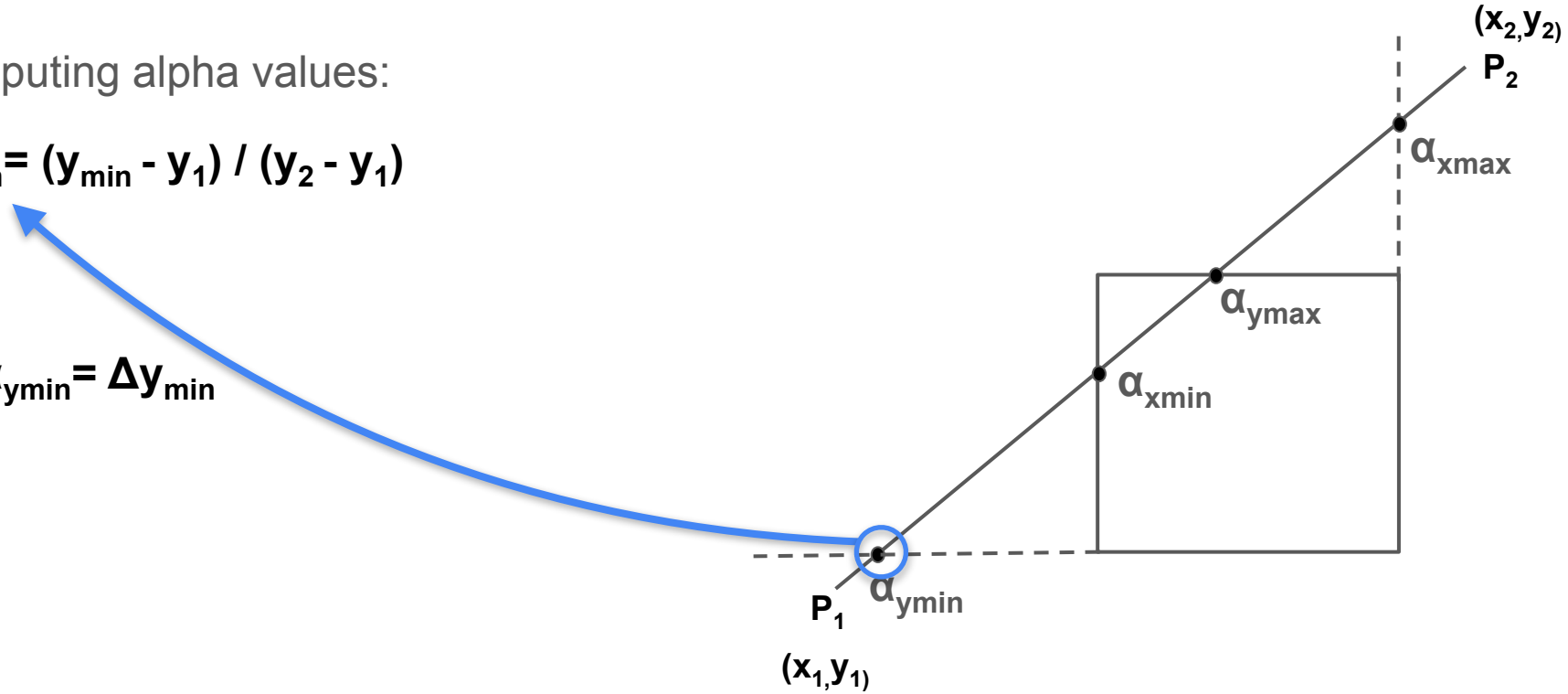
Clipping: Liang-Barsky Line Clipping

Computing alpha values:

$\alpha_{ymin} = (y_{min} - y_1) / (y_2 - y_1)$

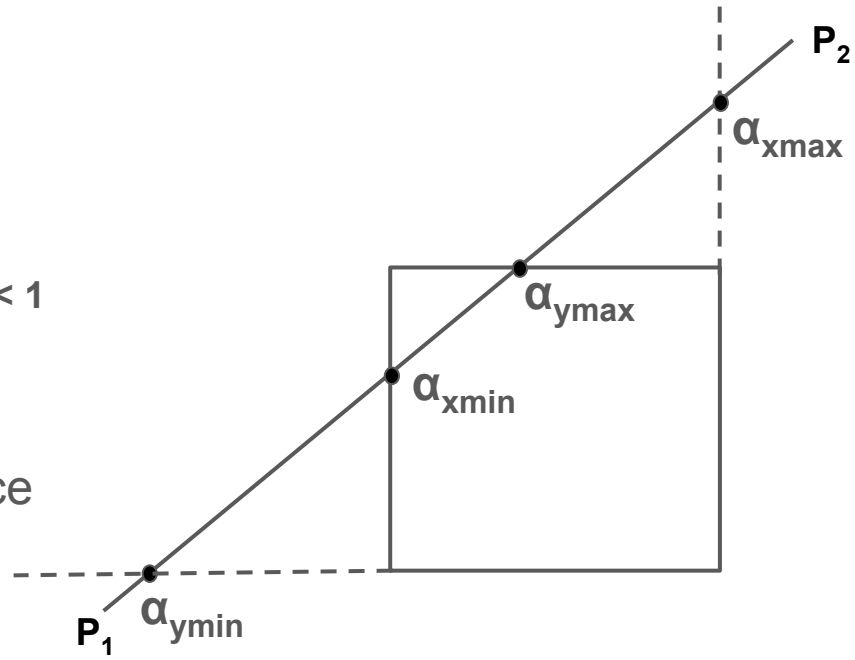
or

$\Delta y\ \alpha_{ymin} = \Delta y_{min}$



Clipping: Liang-Barsky Line Clipping

- Compute the four **α** values corresponding to four sides of the clipping window
- Compare them to find the order.
  - In this case $\mathbf{0 < \alpha_{ymin} < \alpha_{xmin} < \alpha_{ymax} < \alpha_{xmax} < 1}$
- Between two alpha values corresponding to vertical extremes if there is a horizontal alpha value (or vice versa) it indicates the line passes through the clip window.



Clipping: Liang-Barsky Line Clipping

- Any **α** value that is < 0 or > 1 implies the intersection happens on the extension of the actual line segment.